

PLP - Primer Parcial - 1^{er} cuatrimestre de 2019

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido, número de orden y cantidad de hojas en la primera hoja, y numerar las hojas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.

Ejercicio 1 - Programación funcional

En este ejercicio no se permite el uso de recursión explícita, a menos que se indique lo contrario. Pueden usar los ejercicios de la práctica o los vistos en clase, colocando referencias claras. Dar el **tipo** de todas las funciones definidas.

Un **Workspace** está compuesto por varios proyectos. Cada proyecto puede depender de otros proyectos que estén definidos en el mismo **Workspace**. Representamos un **Workspace** en Haskell con una tupla como se muestra a continuación, donde el primer componente es la lista de proyectos definidos (sin repeticiones), y el segundo componente es una función que dado un proyecto p , devuelve una lista de proyectos (sin repeticiones) de los cuales depende p o $[]$ si p no está definido en el **Workspace**.

```
type Project = Int
type Workspace = ([Project], Project -> [Project])
```

Implementar las siguientes funciones.

- hojas, que dado un **Workspace** devuelve los proyectos que no dependen de ningún otro proyecto.
- dependientes, que dado un **Workspace** y un proyecto p devuelve los proyectos que dependen de p .
- fusionar, que dados dos **Workspace** devuelve otro **Workspace** compuesto por los proyectos de ambos. Si un mismo proyecto se encuentra definido en ambos workspaces sus dependencias deben unirse.
- dependencias, que dado un **Workspace** y un proyecto p perteneciente a este, devuelve todas las dependencias transitivas de p (proyectos de los cuales p depende directa o indirectamente). El resultado debe incluir a p . Tener en cuenta posibles dependencias circulares.
Nota: para resolver este punto se permite utilizar recursión explícita.

Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el Cálculo Lambda tipado para soportar colas.

Se extenderán los tipos y términos de la siguiente manera:

$$\sigma ::= \dots \mid \text{Cola}_\sigma \quad M ::= \dots \mid \langle \rangle_\sigma \mid \text{encolar}(M_1, M_2) \mid \text{próximo}(M_1) \mid \text{desencolar}(M_1) \mid \exists x \in M_1/M_2$$

donde $\langle \rangle_\sigma$ es la cola vacía en la que se pueden encolar elementos de tipo σ ; $\text{encolar}(M_1, M_2)$ representa el agregado del elemento M_1 al final de la cola M_2 ; los observadores $\text{próximo}(M_1)$ y $\text{desencolar}(M_1)$ devuelven, respectivamente, el primer elemento de la cola (el primero que se encoló), y la cola sin el primer elemento (estos dos últimos solo tienen sentido si la cola no es vacía); y el observador $\exists x \in M_1/M_2$ indica si existe un elemento x en la cola M_1 que cumpla la condición M_2 , donde M_2 es una expresión booleana que puede contener libre a la variable x .

- Introducir las reglas de tipado para la extensión propuesta.
- Exhibir una derivación del siguiente juicio de tipado:
 $\{c: \text{Cola}_{\text{Nat}}\} \vdash \text{if } \exists x \in c/\text{isZero}(x) \text{ then } \text{próximo}(c) \text{ else } 0: \text{Nat}$
- Definir el conjunto de valores y las nuevas reglas de semántica. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia**, basta con indicar cuántas son.
Pista: puede ser necesario mirar más de un nivel de un término para saber a qué reduce. Por ejemplo, $\text{desencolar}(\text{encolar}(0, \langle \rangle_{\text{Nat}}))$ no reduce de la misma manera que $\text{desencolar}(\text{encolar}(0, \text{encolar}(\underline{1}, \langle \rangle_{\text{Nat}})))$.
- Definir como macro la función esVacía , que indica si la cola pasada como parámetro está vacía (es decir, no tiene elementos encolados).

Ejercicio 3 - Inferencia de Tipos

En este ejercicio extenderemos el algoritmo de inferencia para incorporar el tipado de *grafos* con información en los nodos y en los arcos. El conjunto de tipos y términos es:

$\sigma ::= \dots \mid \text{Grafo}_\sigma$

$M ::= \dots \mid \text{Nil}_\sigma \mid \text{Nodo}(M, N) \mid \text{Arco}(M, N, O, P) \mid \text{case } M \text{ of Nil} \rightsquigarrow N; \text{nodo}(v, g) \rightsquigarrow O; \text{arco}(v_1, v_2, e, h) \rightsquigarrow P$

En el nuevo tipo Grafo_σ , los nodos serán de tipo Nat y los arcos de tipo σ . Utilizaremos Nil_σ para construir un grafo vacío, $\text{Nodo}(M, N)$ para agregar el nodo M al grafo N y $\text{Arco}(M, N, O, P)$ para agregar un arco con valor O entre los nodos M y N en el grafo P . En $\text{case } M \text{ of Nil} \rightsquigarrow N; \text{nodo}(v, g) \rightsquigarrow O; \text{arco}(v_1, v_2, e, h) \rightsquigarrow P$, las variables v, g y v_1, v_2, e, h se ligan en O y en P respectivamente. La reglas de tipado son las siguientes:

$$\frac{\Gamma \triangleright M : \text{Grafo}_\sigma \quad \Gamma \triangleright N : \rho \quad \Gamma \cup \{v : \text{Nat}, g : \text{Grafo}_\sigma\} \triangleright O : \rho \quad \Gamma \cup \{v_1 : \text{Nat}, v_2 : \text{Nat}, e : \sigma, h : \text{Grafo}_\sigma\} \triangleright P : \rho}{\Gamma \triangleright \text{case } M \text{ of Nil} \rightsquigarrow N; \text{nodo}(v, g) \rightsquigarrow O; \text{arco}(v_1, v_2, e, h) \rightsquigarrow P : \rho}$$

$$\frac{}{\Gamma \triangleright \text{Nil}_\sigma : \text{Grafo}_\sigma} \quad \frac{\Gamma \triangleright M : \text{Nat} \quad \Gamma \triangleright N : \text{Grafo}_\sigma}{\Gamma \triangleright \text{Nodo}(M, N) : \text{Grafo}_\sigma} \quad \frac{\Gamma \triangleright M : \text{Nat} \quad \Gamma \triangleright N : \text{Nat} \quad \Gamma \triangleright O : \sigma \quad \Gamma \triangleright P : \text{Grafo}_\sigma}{\Gamma \triangleright \text{Arco}(M, N, O, P) : \text{Grafo}_\sigma}$$

- Dar la extensión al algoritmo necesaria para soportar el tipado de la nueva estructura. **Solo es necesario definir la extensión correspondiente a las primeras tres reglas.** Se considera ya definida la extensión para expresiones de la forma $\text{Arco}(U_1, U_2, U_3, U_4)$.
- Aplicar el algoritmo extendido para tipar la siguiente expresión:

$(\lambda x. \text{case } x \text{ of Nil} \rightsquigarrow v; \text{nodo}(v, g) \rightsquigarrow \text{isZero}(v); \text{arco}(v_1, v_2, e, h) \rightsquigarrow \text{true}) \text{ Nil}$

FECHA 14/05/19

ejercicio
edgardo

1	2	3	T
B	B	B	A

①

type Project = Int

type Workspace = ([Project], Project → [Project])

a) • hojas :: Workspace → [Project]

PODES USA LA
FUNCION NULL

▷ hojas (pp, dep) = filter (λp → dep p == []) pp ✓

b) • dependientes :: Workspace → Project → [Project] ✓

▷ dependientes (pp, dep) p₀ = filter (λp → elem p₀ (dep p)) pp

c) • fusionar :: Workspace → Workspace → Workspace ✓

▷ fusionar (pp₁, dep₁) (pp₂, dep₂) =(concatSinRepe pp₁ pp₂, λp → concatSinRepe (dep₁ p) (dep₂ p))

Donde la auxiliar concatSinRepe es:

• concatSinRepe :: [Project] → [Project] → [Project] ✓

▷ concatSinRepe L₁ L₂ = L₁ ++ (filter (λe → not (elem e L₁)) L₂)

d)

• dependencias :: Workspace → Project → [Project]

▷ dependencias W p = dependenciasDeGrupo W [p]

• dependenciasDeGrupo :: Workspace → [Project] → [Project] ✓

▷ dependenciasDeGrupo (pp, dep) g = if g == (skipDep g dep) then g else dependenciasDeGrupo (pp, dep) (skipDep g dep) ✓

• $\text{stepDep} :: [\text{Project}] \rightarrow (\text{Project} \rightarrow [\text{Project}]) \rightarrow [\text{Project}]$

▷ $\text{stepDep} [] \text{dep} = []$

▷ $\text{stepDep} (p:pp) \text{dep} =$

$\text{concatSimRepe } [p] (\text{concatSimRepe } (\text{dep } p) (\text{stepDep } pp \text{ dep}))$ ✓

$$\textcircled{2} \quad \sigma := \dots \mid \text{Cola}_\sigma$$

$$M := \dots \mid \langle \rangle_\sigma \mid \text{incolar}(M_1, M_2) \mid \text{próximo}(M_1) \mid \text{desincolar}(M_1) \mid \exists x \in M_1 / M_2$$

✓ a)

$$\frac{}{\Gamma \triangleright \langle \rangle_\sigma : \text{Cola}_\sigma} \quad \checkmark \quad \text{T-VACÍA}$$

$$\frac{\Gamma \triangleright M_1 : \sigma \quad \Gamma \triangleright M_2 : \text{Cola}_\sigma}{\Gamma \triangleright \text{incolar}(M_1, M_2) : \text{Cola}_\sigma} \quad \checkmark \quad \text{T-ENCO}$$

$$\frac{\Gamma \triangleright M_1 : \text{Cola}_\sigma}{\Gamma \triangleright \text{próximo}(M_1) : \sigma} \quad \checkmark \quad \text{T-PROX}$$

$$\frac{\Gamma \triangleright M_1 : \text{Cola}_\sigma}{\Gamma \triangleright \text{desincolar}(M_1) : \text{Cola}_\sigma} \quad \checkmark \quad \text{T-DESEN}$$

$$\frac{\Gamma \triangleright M_1 : \text{Cola}_\sigma \quad \Gamma, x : \sigma \triangleright M_2 : \text{Bool}}{\Gamma \triangleright \exists x \in M_1 / M_2 : \text{Bool}} \quad \checkmark \quad \text{T-EX}$$

✓ b) $\{c : \text{Cola}_{\text{Nat}}\} \triangleright \text{if } \exists x \in c / \text{isZero}(x) \text{ then } \text{próximo}(c) \text{ else } 0 : \text{Nat}$

Lemma $\Gamma = \{c : \text{Cola}_{\text{Nat}}\}$

$\frac{c : \text{Cola}_{\text{Nat}} \in \Gamma}{\Gamma \triangleright c : \text{Cola}_{\text{Nat}}} \text{ T-VAR}$

$\Gamma \triangleright c : \text{Cola}_{\text{Nat}}$ ✓

$\frac{x : \text{Nat} \in (\Gamma, x : \text{Nat})}{\Gamma, x : \text{Nat} \triangleright x : \text{Nat}} \text{ T-VAR}$

$\frac{\Gamma, x : \text{Nat} \triangleright x : \text{Nat}}{\Gamma, x : \text{Nat} \triangleright \text{isZero}(x) : \text{Bool}} \text{ T-ISZERO}$

$\frac{\Gamma, x : \text{Nat} \triangleright \text{isZero}(x) : \text{Bool}}{\Gamma \triangleright \exists x \in c / \text{isZero}(x) : \text{Bool}} \text{ T-EX}$

$\Gamma \triangleright \exists x \in c / \text{isZero}(x) : \text{Bool}$ (*) ✓

Falta T-Var

T-PROX $\frac{\Gamma \triangleright c : \text{Cola}_{\text{Nat}}}{\Gamma \triangleright \text{próximo}(c) : \text{Nat}} \text{ T-PROX}$

$\Gamma \triangleright \text{próximo}(c) : \text{Nat}$ ✓

$\frac{}{\Gamma \triangleright 0 : \text{Nat}} \text{ T-ZERO}$

$\Gamma \triangleright 0 : \text{Nat}$ ✓

↓
(*)

T-IF

$\Gamma \triangleright \text{if } \exists x \in c / \text{isZero}(x) \text{ then } \text{próximo}(c) \text{ else } 0 : \text{Nat}$ ✓

c) $V := \dots | \langle \rangle_{\sigma} | \text{encolar}(V, V)$ ✓ (solo aquellos que tipen
no hace falta pedir esto)

$\text{desencolar}(\text{encolar}(V, \langle \rangle_{\sigma})) \rightarrow \langle \rangle_{\sigma}$ ✓

$V_2 \neq \langle \rangle_{\sigma} \forall \text{ tipo}$

$\text{desencolar}(\text{encolar}(V_1, V_2)) \rightarrow \text{encolar}(V_1, \text{desencolar}(V_2))$

$$\exists x \in \langle \rangle_{\sigma} / M \longrightarrow \text{false} \quad \checkmark$$

$$\exists x \in \text{encolar}(V_1, V_2) / M \longrightarrow \text{if } (M\{x \leftarrow V_1\}) \text{ then true} \\ \text{else } \exists x \in V_2 / M \quad \checkmark$$

Supongamos bien definida la sustitución $M\{x \leftarrow N\}$ con esta nueva extensión.

$$\text{próximo}(\text{encolar}(V, \langle \rangle_{\sigma})) \longrightarrow V \quad \checkmark$$

$$V_2 \neq \langle \rangle_{\sigma} \quad \forall \sigma \text{ tipo}$$

$$\text{próximo}(\text{encolar}(V_1, V_2)) \longrightarrow \text{próximo}(V_2)$$

Además de estas, hay 5 reglas de congruencia:

- 2 para encolar \checkmark
- 1 para próximo \checkmark
- 1 para desencolar \checkmark
- 1 para $\exists x \in M / N$ \checkmark

✓d) esVacia c \equiv if Exec/true then false else true ✓
esVacia = nc. ---

③

$$\sigma := \dots \mid \text{Grafo}_\sigma$$

$$M := \dots \mid \text{Nil}_\sigma \mid \text{Nodo}(M, N) \mid \text{Anco}(M, N, O, P)$$

$$\mid \text{Case } M \text{ of } \text{Nil} \rightarrow N; \text{nodo}(r, g) \rightarrow O; \text{anco}(r_1, r_2, s, h) \rightarrow F$$

a)

1) $\bullet W(\text{Nil}) \stackrel{\text{def}}{=} \emptyset \triangleright \text{Nil}_s: \text{Grafo}_s$ con "s" fresca

2) Si

$$W(U) = \Gamma_1 \triangleright M: \sigma_1$$

$$W(V) = \Gamma_2 \triangleright N: \sigma_2$$

$$\text{Sea } S = \text{MGU}(\{ \tau_1 \doteq \tau_2 \mid x: \tau_1 \in \Gamma_1 \wedge x: \tau_2 \in \Gamma_2 \} \cup \{ \sigma_1 \doteq \text{Not}, \tau \doteq \text{Grafo}_s \})$$

con "s" fresca

$$\Rightarrow \bullet W(\text{Nodo}(U, V)) \stackrel{\text{def}}{=} S\Gamma_1, U S\Gamma_2 \triangleright \text{Nodo}(SM, SN): S \text{Grafo}_s$$

(Para todo el siguiente supongo dada la extensión para Grafos del MGU)

3) Si

$$W(U) = \Gamma_1 \triangleright M: \sigma_1$$

$$W(V) = \Gamma_2 \triangleright N: \sigma_2$$

$$W(W) = \Gamma_3 \triangleright O: \sigma_3$$

$$W(X) = \Gamma_4 \triangleright P: \sigma_4$$

Si Γ_3 tuviera información de tipos sobre r o g ,
se los quito en Γ'_3 :

$$\Gamma'_3 = \Gamma_3 \setminus \{r:\pi_1, g:\pi_2\} \quad (\text{abuso de notación, aclaro abajo})$$

Si Γ_4 tuviera sobre r_1, r_2, e o h , se la quito en Γ'_4 :

$$\Gamma'_4 = \Gamma_4 \setminus \{r_1:\pi_3, r_2:\pi_4, e:\pi_5, h:\pi_6\} \quad (\text{abuso de notación, aclaro abajo}).$$

Si tuviéramos solo información parcial, los quito toda la que
tenemos. Más formalmente, Γ'_3 y Γ'_4 son Γ_3 y Γ_4 respect.
pero sin información de tipos sobre las variables ' r ' y ' g ', en Γ'_3
y ' r_1, r_2, e, h ' en Γ'_4 .

Sea:

$$S = \text{MGU}(\{ \tau_1 \doteq \tau_2 \mid x:\tau_1 \in \tilde{\Gamma}_1 \wedge x:\tau_2 \in \tilde{\Gamma}_2 \text{ con } \tilde{\Gamma}_1, \tilde{\Gamma}_2 \in \{\Gamma_1, \Gamma_2, \Gamma'_3, \Gamma'_4\} \} \\ \cup \{ \sigma_1 \doteq \text{Grabo}_S, \sigma_2 \doteq \sigma_3, \sigma_3 \doteq \sigma_4 \}) \text{ con "S" fresca}$$

Si hubiera un $r:\pi_1 \in \Gamma_3$, agrego a S $\{\pi_1 \doteq \text{Nat}\}$

Si hubiera un $g:\pi_2 \in \Gamma_3$, agrego a S $\{\pi_2 \doteq \text{Grabo}_S\}$

Si hubiera un $r_1:\pi_3 \in \Gamma_4$, agrego a S $\{\pi_3 \doteq \text{Nat}\}$

Si hubiera un $r_2:\pi_4 \in \Gamma_4$, agrego a S $\{\pi_4 \doteq \text{Nat}\}$

Si hubiera un $e:\pi_5 \in \Gamma_4$, agrego a S $\{\pi_5 \doteq S\}$

Si hubiera un $h:\pi_6 \in \Gamma_4$, agrego a S $\{\pi_6 \doteq \text{Grabo}_S\}$

$$\Rightarrow \bullet W(\text{case } U \text{ of Nil} \rightarrow V; \text{modo}(r,g) \rightarrow W; \text{anco}(r_1, r_2, e, h) \rightarrow X)$$

$$= S\Gamma_1 U S\Gamma_2 U S\Gamma'_3 U S\Gamma'_4 \triangleright \text{case } S M \text{ of Nil} \rightarrow S N; \text{modo}(r,g) \rightarrow S O; \\ \text{anco}(r_1, r_2, e, h) \rightarrow S P : S \sigma_2$$

$$S = \text{MGU}\{ \text{Grabs}_{s_4} \rightarrow \text{Bool} \rightarrow s_5 \}$$

$$b) \left[\{v:\text{Bool}\} \triangleright (\lambda x:\text{Grabs}_{s_5} \text{ case } x \text{ of Nil} \rightarrow v; \text{Node}(v,g) \rightarrow \text{isZero}(v); \text{and}(\sigma_1, \sigma_2, e, h) \rightarrow \text{true}) \text{Nil} \right]_{s_5}$$

↓ APP

:Bool

$$\left[\{v:\text{Bool}\} \triangleright \lambda x:\text{Grabs}_{s_5} \text{ case } x \text{ of Nil} \rightarrow v; \text{Node}(\sigma g) \rightarrow \text{isZero}(v); \text{and}(\sigma_1, \sigma_2, e, h) \rightarrow \text{true} : \text{Grabs}_{s_5} \right] \rightarrow \text{Bool}_{s_5}$$

↓ ABS

$$S = \text{MGU}\{ s_1 \doteq \text{Grabs}_{s_3}, s_2 \doteq \text{Bool}, \text{Bool} \doteq \text{Bool}, \text{Nat} \doteq \text{Nat} \}$$

$$\left[\{v:\text{Bool}, x:\text{Grabs}_{s_3}\} \triangleright \text{case } x \text{ of Nil} \rightarrow v; \text{Node}(\sigma g) \rightarrow \text{isZero}(v); \text{and}(\sigma_1, \sigma_2, e, h) \rightarrow \text{true} : \text{Bool} \right]_{s_5}$$

CASE

$$\left[\{v:s_2\} \triangleright v:s_2 \right]$$

$$\left[S = \text{MGU}\{ s_3 \doteq \text{Nat} \} \{v:\text{Nat} \text{isZero}(\sigma v) : \text{Bool} \} \right]$$

CASE

$$\left[\emptyset \triangleright \text{true} : \text{Bool} \right]$$

$$\left[\{x:s_1\} \triangleright x:s_1 \right]$$

$$\text{isZero}$$

$$\left[\{v:s_3\} \triangleright v:s_3 \right]$$

(Nota: los Si se refieren siempre a variables frescas, incrementando "n" en los nombres)